REMARKS

In an Office Action dated June 4, 2004, the Examiner objected to the Drawing; rejected claims 1, 4, 9, 12-14, 17, 20, 22, 24, 29, and 32-34 under 35 U.S.C. 112, second paragraph, as indefinite; and rejected claims 1-36 under 35 U.S.C. 103(a) as obvious over Johnson ("XML for the absolute beginner") in view of Meltzer et al. (U.S. Patent 6,125,391) .

Objection to the Drawing

The Examiner objected to the Drawing for failure to show the recited ""transformation language" of claims 6 and 26, and "XSL" of claims 7, 16, 27 and 36. Applicants respectfully traverse this objection.

Claims 6 and 26 recite that the "code generation template" recited in the independent claims is "implemented in a transformation language". A transformation language is not a recited feature of applicant's invention. The recited feature is the "code generation template", which is indeed shown in Fig. 1 as feature number 16. Being "implemented in a transformation language" is simply an attribute of the recited "code generation template".

Applicants are frankly at a loss to understand how the Examiner proposed to represent "implemented in a transformation language" in the Drawing. The "code generation template" is clearly shown as block 16, but it is still a code generation template, and the drawing representation is the same, whether it is "implemented in a transformation language" or not. It would be simply misleading to show a "transformation language" as a separate feature in the drawing, for that may imply it is part of applicant's invention herein. According to claims 6 and 26, a transformation language is simply the way in which the code generation template is implemented, i.e., the form of the code generation template. It is an attribute of feature 15, not a

separate claimed feature of the invention. Applicants do not recite a "transformation language" as an element of any claim..

The same reasoning applies to the recitations of claims 7 and 27 (claims 16 and 26 have been cancelled). These claims recite in greater particularity that the transformation language is XSL, but again, the recited feature is the code generation template, and being implemented in XSL is simply an attribute of that recited feature.

Indefiniteness Rejections

The Examiner maintains his objection to the use of the trademark "JAVA" in various claims. Applicants respectfully traverse the rejection. JAVA is a programming language and environment originally developed by Sun Microsystems, which retains certain trademark rights in the name JAVA. However, the semantic specification of the programming language, and requirements for associated environmental entities such as JAVA compilers, are publicly available, and are well known. The claims do not recite a source of the goods. I.e., there is nothing in the claims that requires the recited JAVA classes or other elements to originate from Sun Microsystems, or from any other particular source. The Examiner contrasts JAVA with C++, stating that products incorporating the name C++ originate from many sources. Applicants respectfully submit that the same is true of JAVA. Sun has granted the whole world a license to write code in the JAVA language and to identify it as such, provided that the code complies with the semantic definition of the JAVA language. The fact that JAVA was originally created by a private corporation is not dispositive. Applicants would point out that many programming languages originated in private corporations. For example, FORTRAN, the granddaddy of all high-level languages, was originally an IBM product. Nor does the fact that the JAVA specification might undergo occasional revision render the claims indefinite. All programming languages evolve over time. The same might be said of the English language. Are we to take the

position that, because the meaning of English words evolves over time, claims written in the English language are *per se* indefinite?

The ultimate question is simply this: given a claim recitation of a "JAVA class", is it possible to determine with reasonable particularity whether an accused method or device falls within the scope of the claim? Applicants submit that the answer is manifestly "yes". Thousands of programmers write code in the JAVA language every day. These are people of ordinary skill in the art, and presumably they are able to tell what is a JAVA class and what isn't, or they wouldn't be able to perform their jobs.

The Examiner further objected to the lack of an essential structural cooperative relationship of elements in claims 1, 9, 13, 17, 22, 29 and 33. Claims 13 and 33 are cancelled, and the rejection thereof is moot. Applicants have amended the remaining claims in a manner believed to recite sufficient cooperative relationship. Specifically, the claims are amended to recite that code embodying the data access class (or data access JAVA class) is generated as output from the code generation template applied to the data object description document or data object descriptor or XML data object description, as the case may be. These amendments resolve any failure to recite the structural cooperative relationships, and applicants submit that the claims, as amended, are sufficiently definite.

Prior Art

Applicants have cancelled claims 2, 10, 13-16, 18, 23 , 30 and 33-36, and the rejection thereof is moot. Applicants have amended the remaining independent claims to more specifically recite their invention. In particular, the independent claims have been amended to recite that the data object description document (or analogous entity) conforms to a data object document type definition defining at least one API for accessing backend data at runtime, that the code

generation template includes an API call for the AP, and that the generated data access class uses the API call. As amended, the claims are patentable over the cited art.

Applicants' invention is intended as an improved technique for generating and maintaining data access code in an object-oriented environment, i.e., data access classes which access data in a database. This is conventionally accomplished by manually writing special purpose classes to invoke defined APIs for accessing the data. Because many classes may need to access the same data, this technique can result in inconsistencies, is error-prone and time consuming. In accordance with applicants' invention, a data object description for the existing data model is created (preferably in XML). A data access code generator applies code generation templates to the data object description to automatically generate code. The template includes the calls to the APIs for accessing the data, and the resultant automatically generated code uses these API calls. By means of this technique, a single data object description might be used in conjunction with different code generation templates to create different data access classes for accessing different data using different APIs. Alternatively, multiple data object descriptions can use a common code generation template to access the same data on behalf of different programs or processes. Similarly, changes can be made to the data object description and propagated to the data access objects, without changing the templates.

Therefore a key feature of applicant's invention is the generation of *code* (specifically a data access class in an object-oriented environment) which *accesses data using an API.*

Applicants' representative amended claim 1 recites:

1. A method of data access code generation in an object-oriented programming environment, comprising:

a) describing a data object in a data object description document, said data object description document conforming to a data object document type definition, said data object document type definition *defining at least one application programming interface (API) for providing access to backend data at runtime*;

b) applying at least one code generation template to said data object description document, *said at least one code generation template including at least one data access API call* for at least one corresponding API for providing access to backend data at runtime, said API defined by said data object document type definition;

c) generating *code* in an object-oriented programming language, said *code* embodying at least one data access class to provide access to said backend data at runtime *using said at least one data access API call*, said code being generated automatically as output from said at least one code generation template applied to said data object description document. [emphasis added]

Johnson discloses that XSL rules ("templates") can be applied to an input XML file to produce a new structure. However, what is being produced is simple text. I.e., the example of *Johnson* is the translation of an XML file to HTML. using an XSL file as a set of templates.

Meltzer discloses that a document in XML format is translated to a different format, specifically a JAVA object which carries the data of the XML document. The purpose of the translation is to permit the document to be accessed in a different system environment based on JAVA.

Although *Meltzer* refers to JAVA objects, a careful reading shows that *Meltzer* is translating what is essentially textual data from XML to a JAVA object. I.e., the XML document is a business form (purchase order, invoice or the like), and the JAVA object is a form of container for the XML document data in an object-oriented environment. The JAVA object thus includes methods (e.g., get and set) for accessing the data in the JAVA object. However, this is not a data access class as that feature is used and defined in applicants' claims. Specifically, as

recited in applicants' claims, the data access class uses an API call to access backend data (outside the object) at runtime.

Thus, *Meltzer* is similar to *Johnson* in that the translation is essentially from one form of text data to another. In an object-oriented environment, it is convenient to maintain data (including text data) in an object which includes methods for accessing the data. *Meltzer* does not disclose that these methods are generated from the XML document, and they presumably are separately defined in the target system. By translating the XML document to a JAVA object, it is convenient for the existing methods in the target system to operate on the data in the XML document.

Together, the combination of *Johnson* and *Meltzer* might conceivably suggest that the translation mechanism of *Johnson* be used to translate an XML document to a JAVA object as in *Meltzer*. Applicants' claims do indeed include a translation mechanism as in *Johnson* for translating XML documents, but there the similarity ends. The XML document is a definition of a data access class, not a business form as in *Meltzer*. The template includes an API call for accessing backend data at runtime, rather than a set of rules for placing the business form data in a JAVA object. The result of the translation is a data access class which uses the API call for assessing backend data at runtime. *Meltzer* discloses only that standard methods would be included in the JAVA object for accessing data (i.e., the business form) within the object itself.

For all of these reasons, the proposed combination of *Johnson* and *Meltzer* does not teach or suggest applicant's claimed invention, as amended.

In view of the foregoing, Applicants submit that the claims are now in condition for allowance and respectfully request reconsideration and allowance of all claims. In addition, the Examiner is encouraged to contact applicants' attorney by telephone if there are outstanding issues left to be resolved to place this case in condition for allowance.

Respectfully submitted,

DAVID B ORCHARD, et al.

By:_____
Roy W. Truelson
Registration No. 34,265

Telephone: (507) 289-6256